EECE5639
COMPUTER VISION

PROJECT 1

MOTION DETECTION USING SIMPLE IMAGE FILTERING

SHIVA KUMAR DANDE
SAI PRASASTH KOUNDINYA GANDRAKOTA

ABSTRACT

In this project we aimed to optimize the motion detection algorithm by examining various filters and parameters. Specifically, we compared the effectiveness of using Gaussian and temporal differential filters in capturing changes between video frames, as well as the impact of spatial filters (e.g., 3x3 Box, 5x5 Box, Gaussian) on the algorithm's performance. Additionally, we explored the role of standard deviation in the frame derivatives and determined the optimal threshold for this parameter to minimize the number of false positives and negatives.

DESCRIPTION OF ALGORITHMS

1D TEMPORAL DERIVATIVE FILTERS

Two types of temporal derivative filters were used to filter the image frames to detect motion: a Differential Operator and a 1D Derivative of a Gaussian (with user-defined Standard Deviation).

Each filter was applied to a set of three sequential image frames and then a mask was created from the merged image frames. The mask is then applied to each image frame to give us our desired result.

The differential operator results in an image showing the difference between two image frames thus detecting motion. The 1D Derivative of a Gaussian gives us an output with a lot of noise in the center of the image which was plotted with the user-defined standard deviation.

2D SPATIAL SMOOTHENING FILTERS

In order to reduce the noise, we then convolved the image frames with a 2D Box Filter and a 2D Gaussian Filter before applying the temporal derivative filter and obtaining a differential mask and a Gaussian mask.

The 2D Box Filter and 2D Gaussian filter resulted in burred but smoother images, reducing the noise and allowing easier detection of motion. As the size of the filter grew the boundaries got sharpened and larger filtered subsets are visible thus detecting in smoother motion detection.

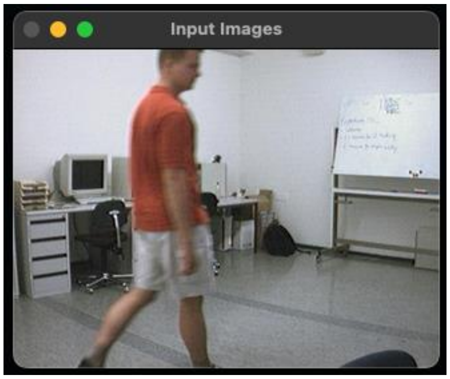
THRESHOLDING

To isolate the regions of interest, it is necessary to establish a threshold value that filters out insignificant variations from frame to frame. We tested different threshold values to create a 0 and 1 mask of the moving objects.

EXPERIMENTS

INPUTS

ORIGINAL IMAGE



A sequence of 353 image frames were read into the program with various levels of motion in several of them. Our main focus was the image frames where certain movement (i.e., the red chair, person 1 walking through the frame, person 2 walking into the frame and the light being switched on, etc.).

GREYSCALE IMAGE

Each of the 353 image frames were then converted to greyscale in order to observe intensity values at a pixel over time which gives us a stable/slowly varying signal, except when an object in motion begins to shadow that pixel, in which case the intensity of the foreground object takes over from the intensity of the background pixel.

OUTPUTS

WITHOUT SMOOTHENING

```
Enter the Threshold value : 3
No of Images in the given path is 353
Enter the sigma Value : 1.4
```

Masks

Differential and gaussian masks

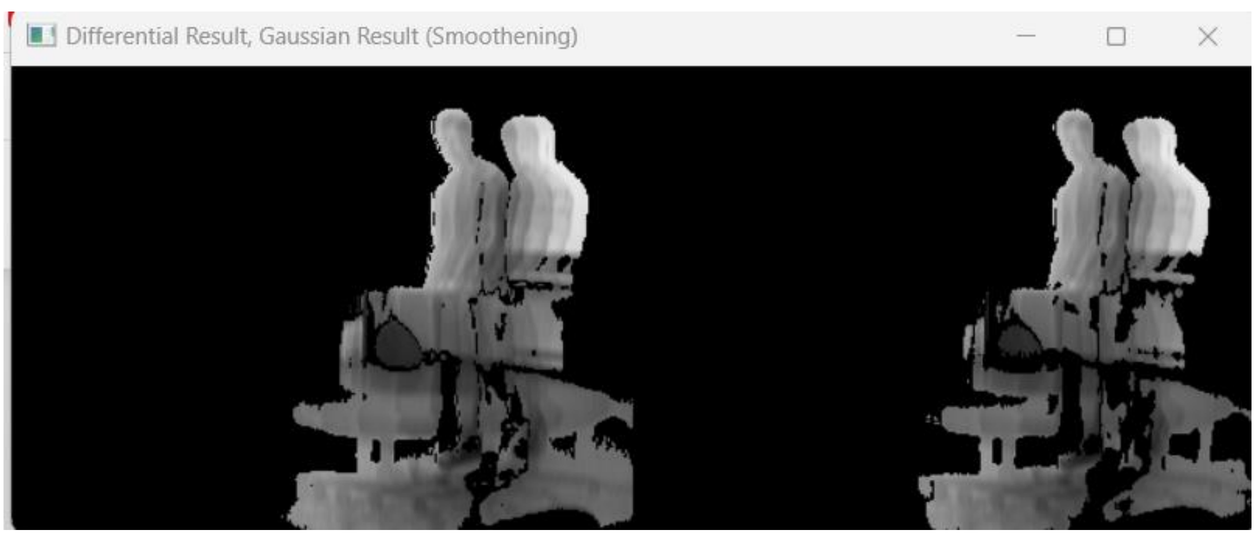
Result

Differential and gaussian results

WITH SMOOTHENING

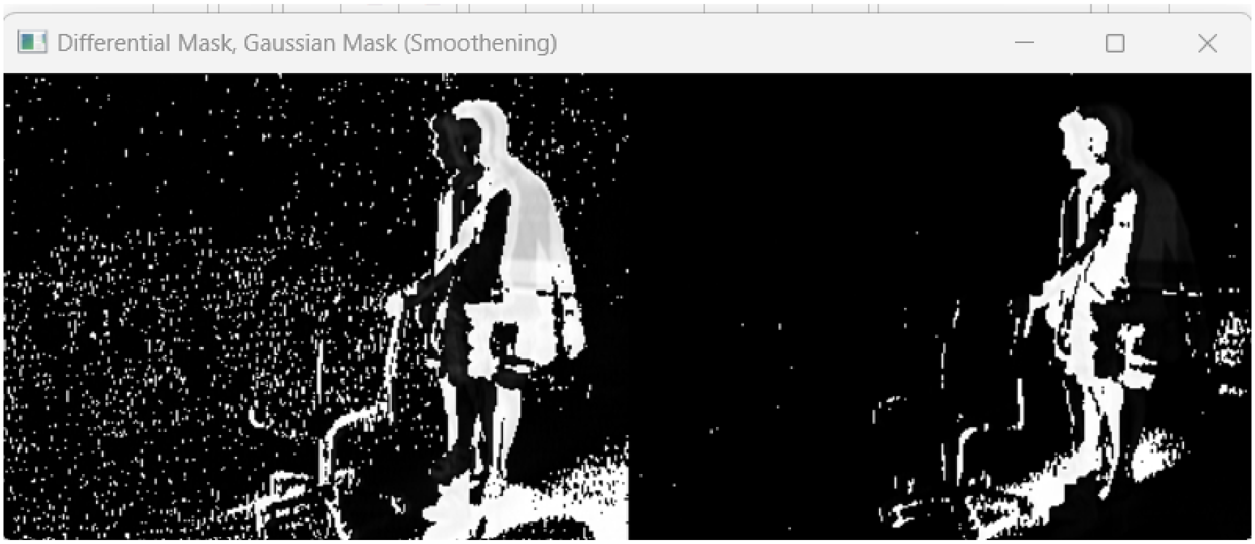
5X5 BOX FILTER

Differential and gaussian masks

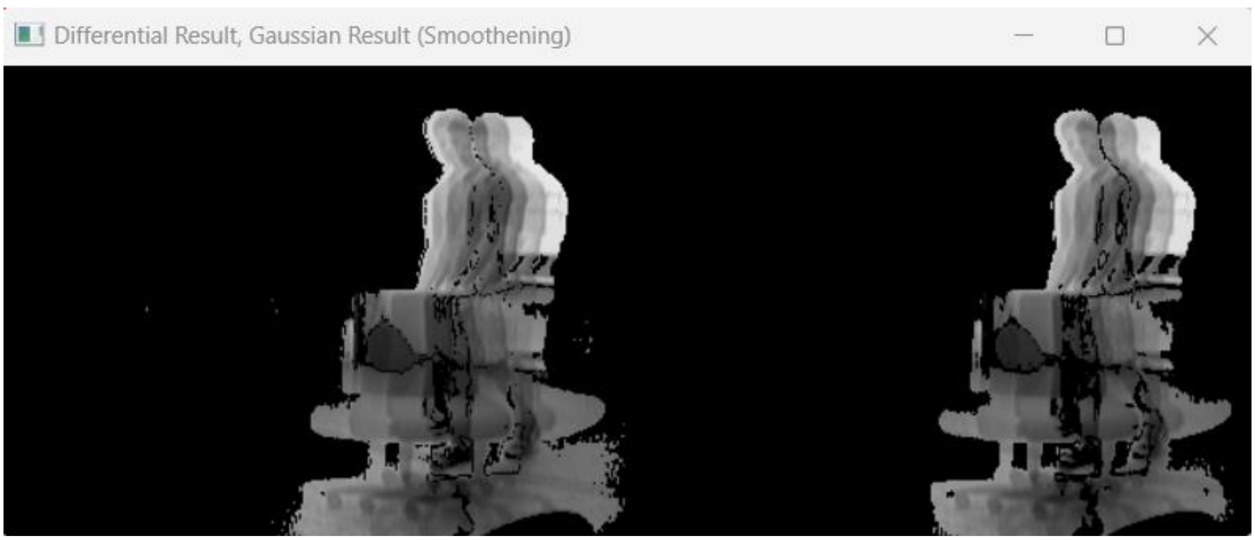


Differential and gaussian results

3X3 BOX FILTER

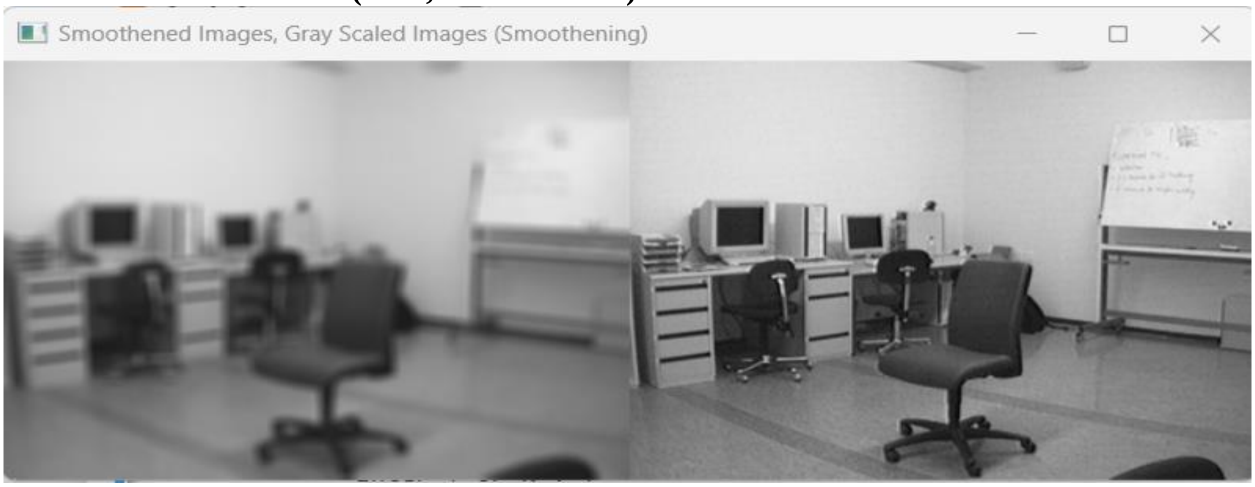


Differential and gaussian masks

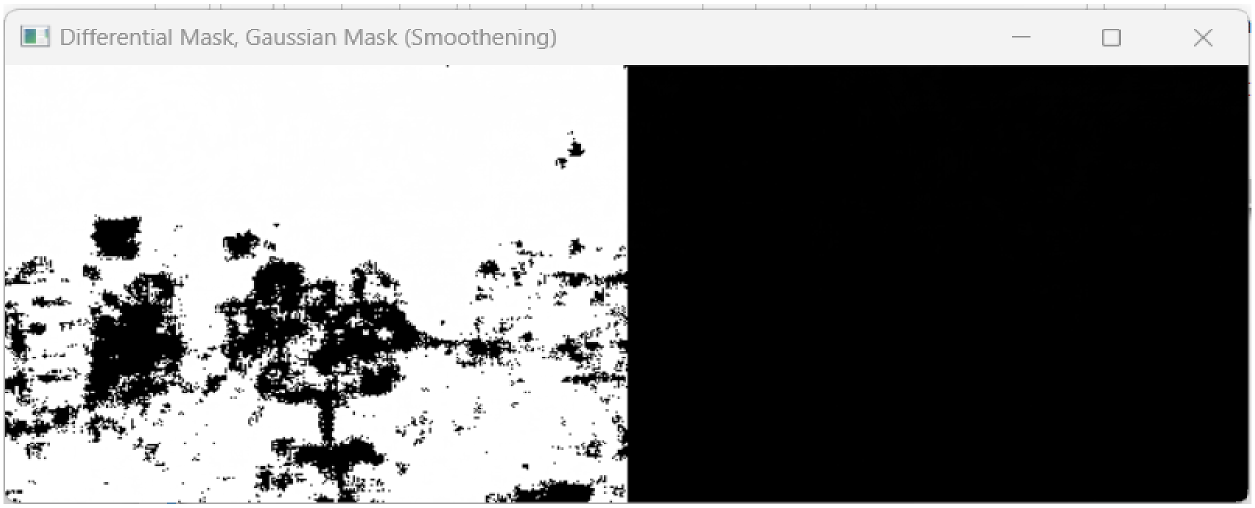


Differential and gaussian results

2D GAUSSIAN FILTER (7X7 ; SIGMA = 1.4)



Smoothened Image using 2D gaussian filter

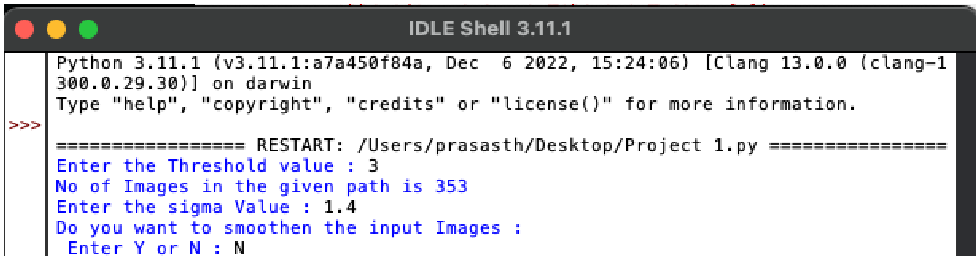


Differential and gaussian masks

Differential and gaussian results

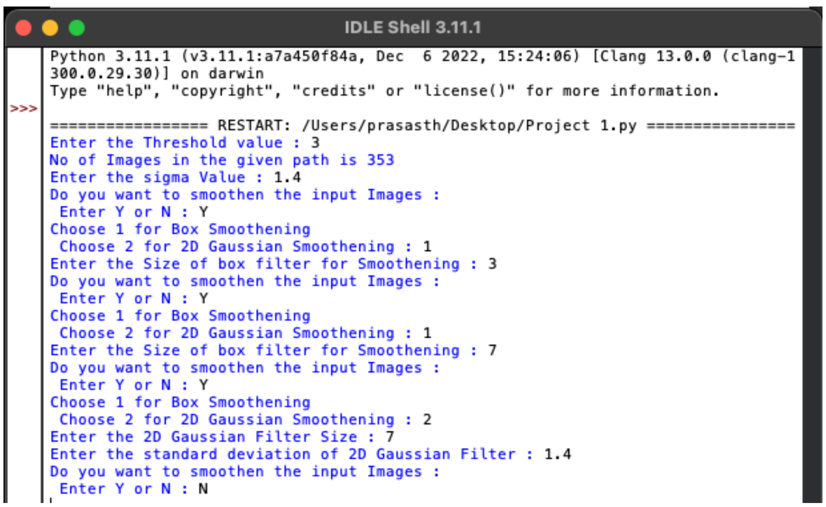
PARAMETERS USED

In this project the above experiments were all done with user-defined threshold and standard deviation values (3 and 1.4 respectively). For the spatial smoothening filters, the size of the box filters and the 2D Gaussian filter were user-defined as well.



```
IDLE Shell 3.11.1
Python 3.11.1 (v3.11.1:a7a450f84a, Dec  6 2022, 15:24:06) [Clang 13.0.0 (clang-1
300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: /Users/prasasth/Desktop/Project 1.py ================
Enter the Threshold value : 3
No of Images in the given path is 353
Enter the sigma Value : 1.4
Do you want to smoothen the input Images :
 Enter Y or N : N
```

The differential operator filter was taken as 0.5*[-1,0,1] which resulted in the differential output for the filtered images.



```
IDLE Shell 3.11.1
Python 3.11.1 (v3.11.1:a7a450f84a, Dec  6 2022, 15:24:06) [Clang 13.0.0 (clang-1
300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: /Users/prasasth/Desktop/Project 1.py ================
Enter the Threshold value : 3
No of Images in the given path is 353
Enter the sigma Value : 1.4
Do you want to smoothen the input Images :
 Enter Y or N : Y
Choose 1 for Box Smoothening
 Choose 2 for 2D Gaussian Smoothening : 1
Enter the Size of box filter for Smoothening : 3
Do you want to smoothen the input Images :
 Enter Y or N : Y
Choose 1 for Box Smoothening
 Choose 2 for 2D Gaussian Smoothening : 1
Enter the Size of box filter for Smoothening : 7
Do you want to smoothen the input Images :
 Enter Y or N : Y
Choose 1 for Box Smoothening
 Choose 2 for 2D Gaussian Smoothening : 2
Enter the 2D Gaussian Filter Size : 7
Enter the standard deviation of 2D Gaussian Filter : 1.4
Do you want to smoothen the input Images :
 Enter Y or N : N
```

The box filters were of sizes 3x3 and 7x7 respectively for the above experiments, and for the 2D Gaussian filter the size was taken on basis of the user given standard deviation value, which in this case was m = 7 (m >= 5*1.4 ➔ m>=7).

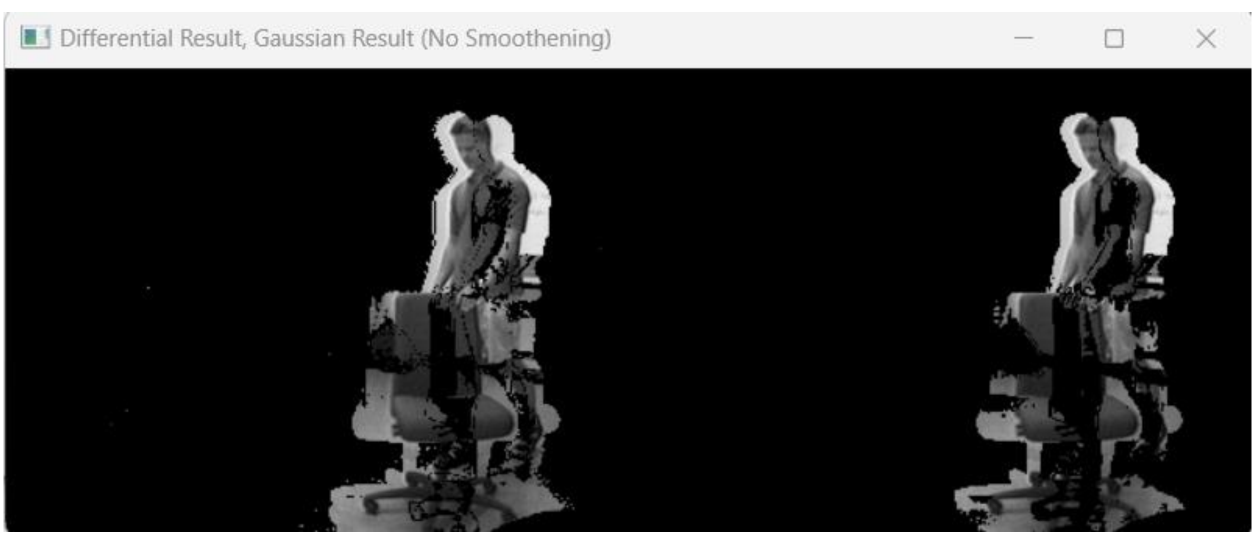
OBSERVATIONS AND CONCLUSIONS

A significant constraint in our current method of identifying the region of interest lies in the fact that the mask only records the position of the moving component in the preceding and succeeding frames and fails to account for the movement of the object in the current frame. Consequently, when the moving component moves at a rapid pace, the two bounding regions become disjointed, creating a gap in the area where the actual object (person) is located. This is because the dissimilar pixels between the two frames signify the starting and ending point of the object in each frame. Therefore, rather than highlighting motion through distinct windows, the variations can be utilized as boundary edges of the window.

VARYING THRESHOLD VALUES

```
Enter the Threshold value : 10
No of Images in the given path is 353
Enter the sigma Value : 1.4
```

Differential and gaussian masks



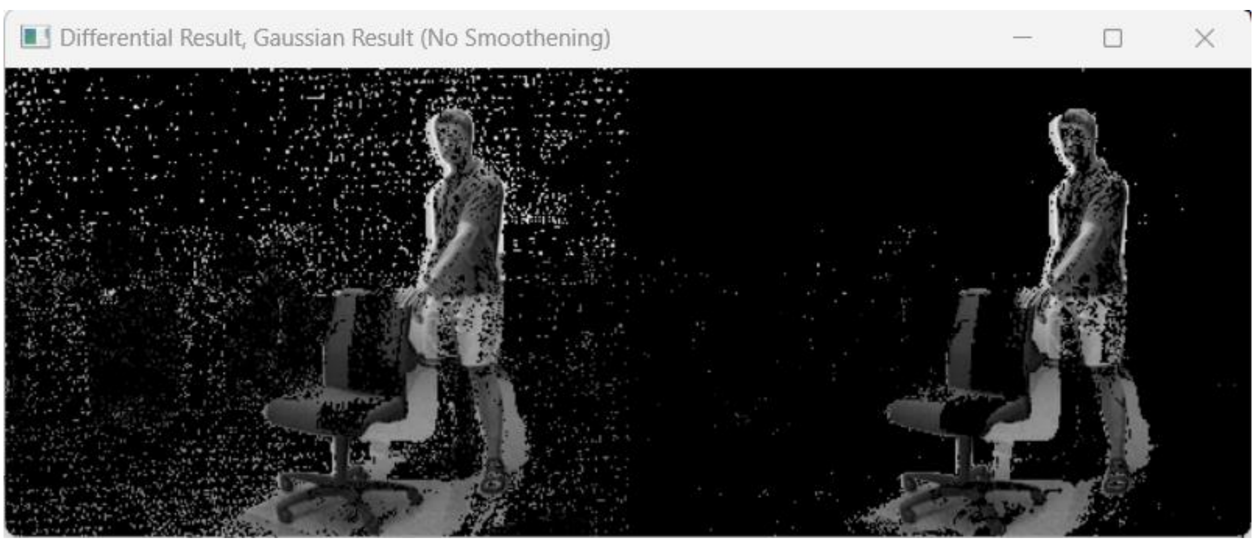
Differential and gaussian results

As we vary the threshold for the masks, we observe that when the threshold is set to a fixed value, large changes in illumination cause the entire image frame to be considered as motion. Upon varying the threshold like shown above we see the differences between masks and results for different threshold values which can allow us to determine if any results of pixel changes are caused by external factors or if the changes in pixel values are expected.

VARYING SIGMA VALUES

```
Enter the Threshold value : 3
No of Images in the given path is 353
Enter the sigma Value : 2.4
```

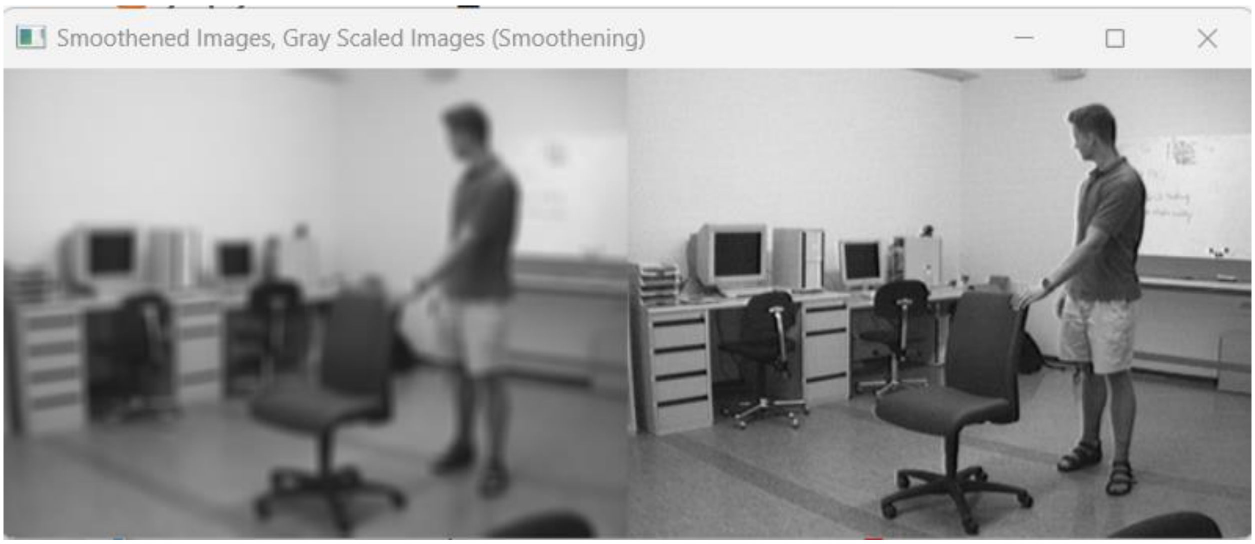
Differential and gaussian masks



Differential and gaussian results

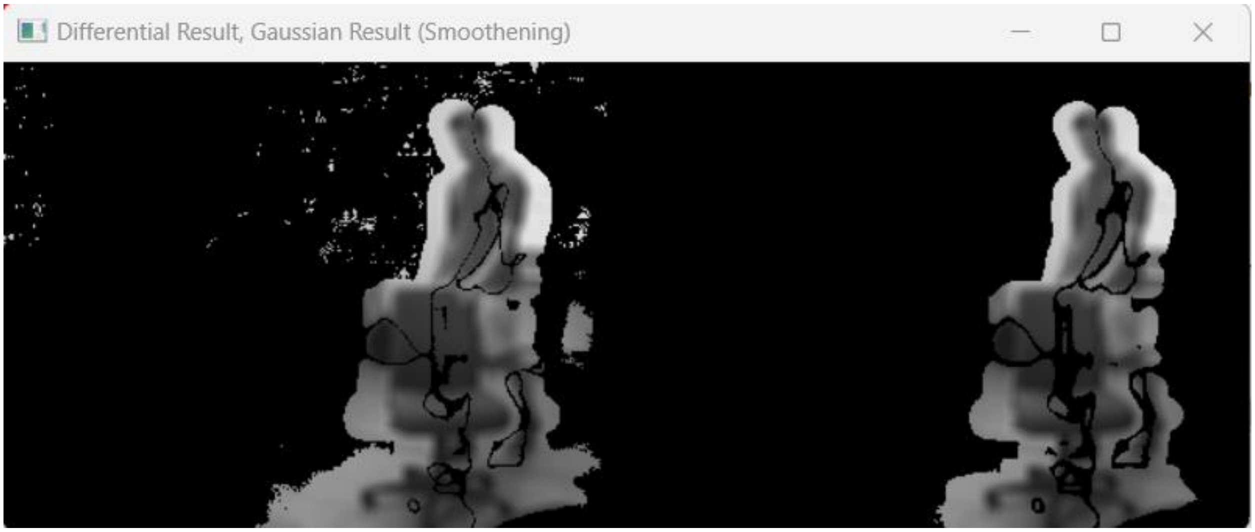
For the above results we varied the value of the standard deviation of the temporal Gaussian mask which shows us that a higher value of sigma adds more noise to the outcome of applying the mask. This is rectified partially by applying a smoothening filter before we apply the temporal filter. As we can see in the results below, the smoothened images lead to a reduction in noise in the masks and the resulting images.

CONCLUSIONS



Smoothened image (Gaussian 2D) and original gray sclaed image

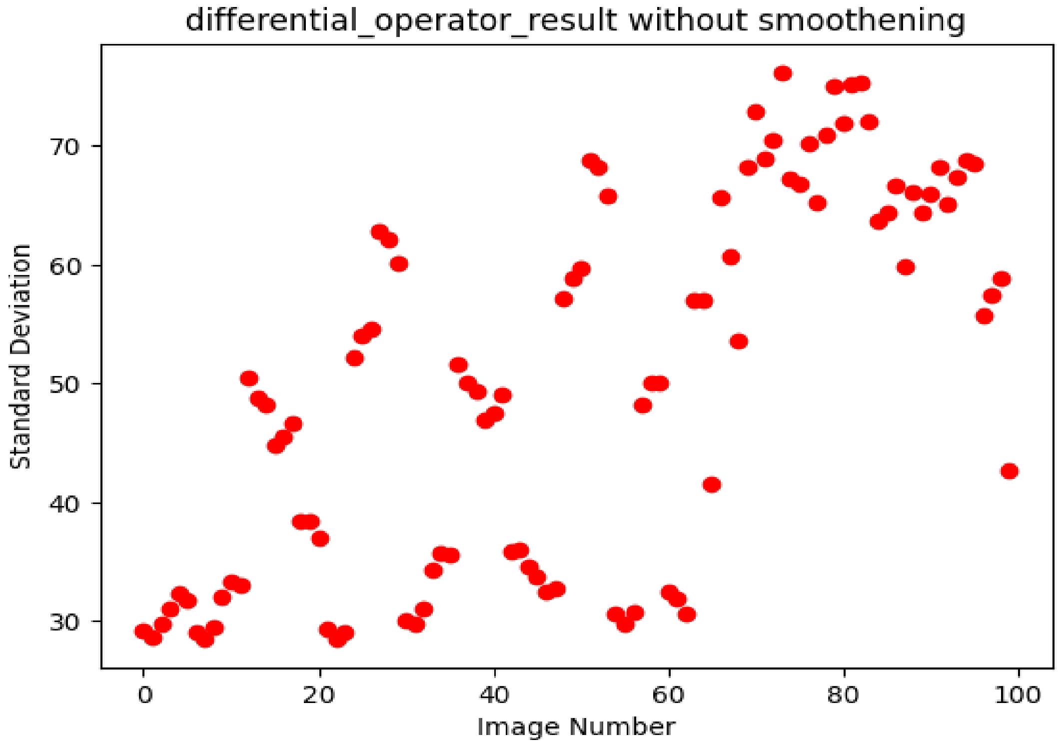
Differential and gaussian masks

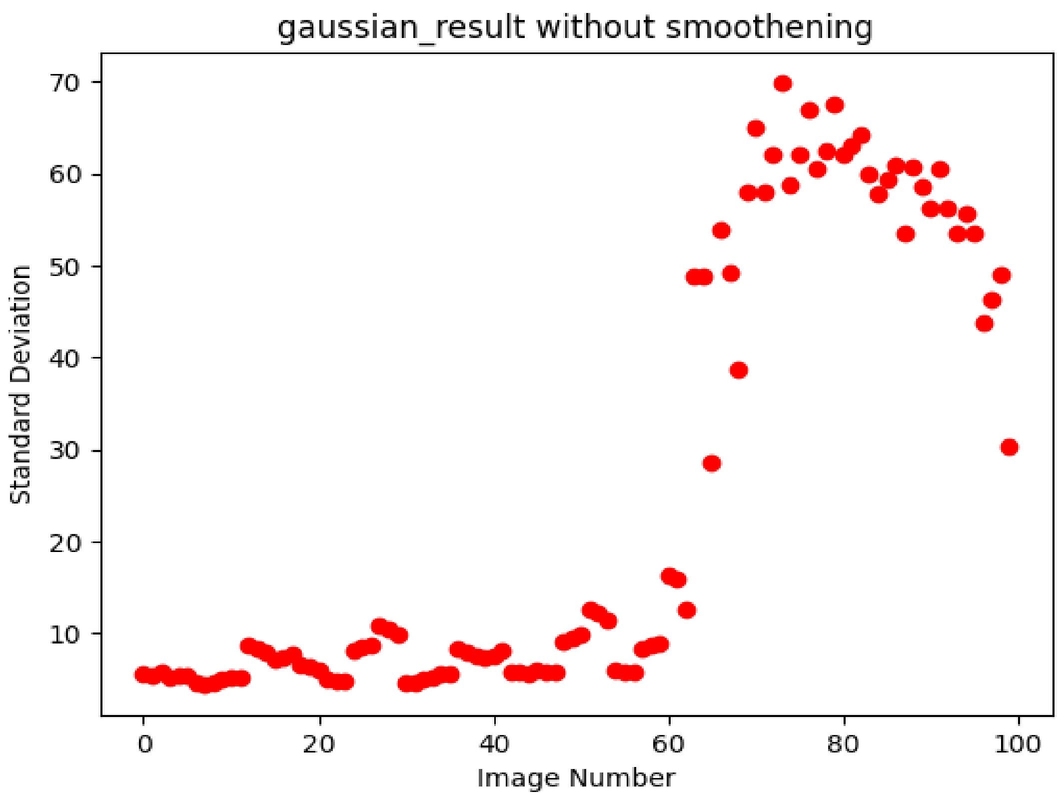


Differential and gaussian results

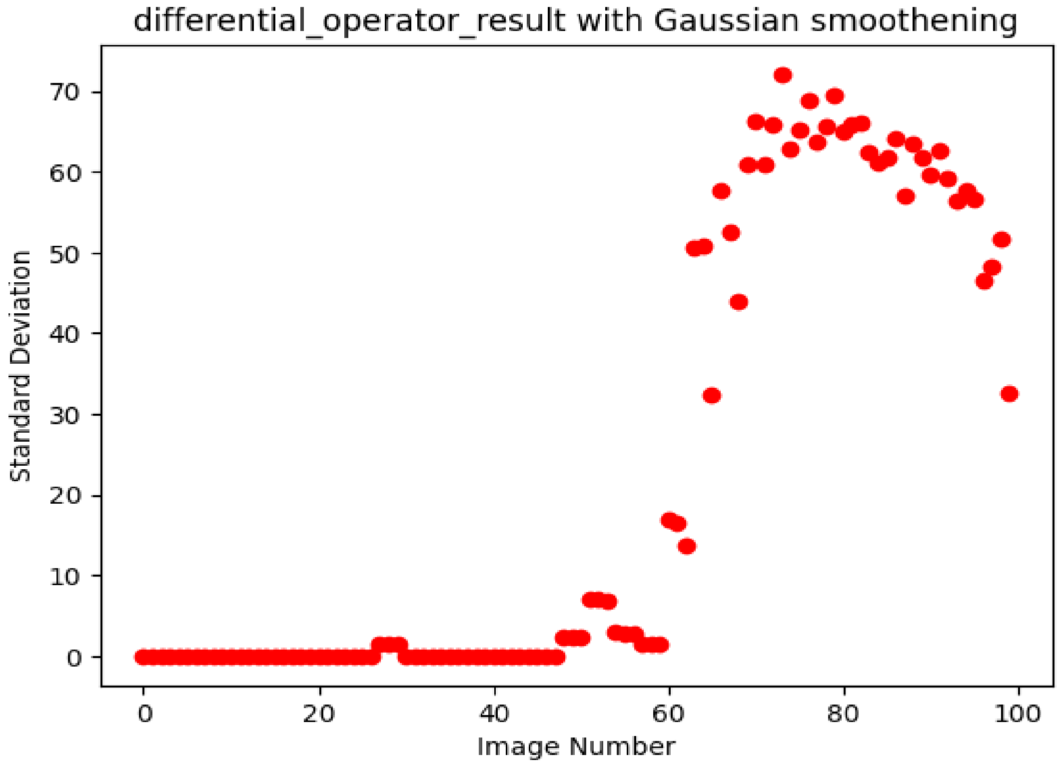
Finally, we conclude that motion detection using simple image filtering can be optimized by implementing a spatial smoothening filter before applying the temporal derivative mask along with varying values of the threshold and standard deviation can help us detect movement between frames.

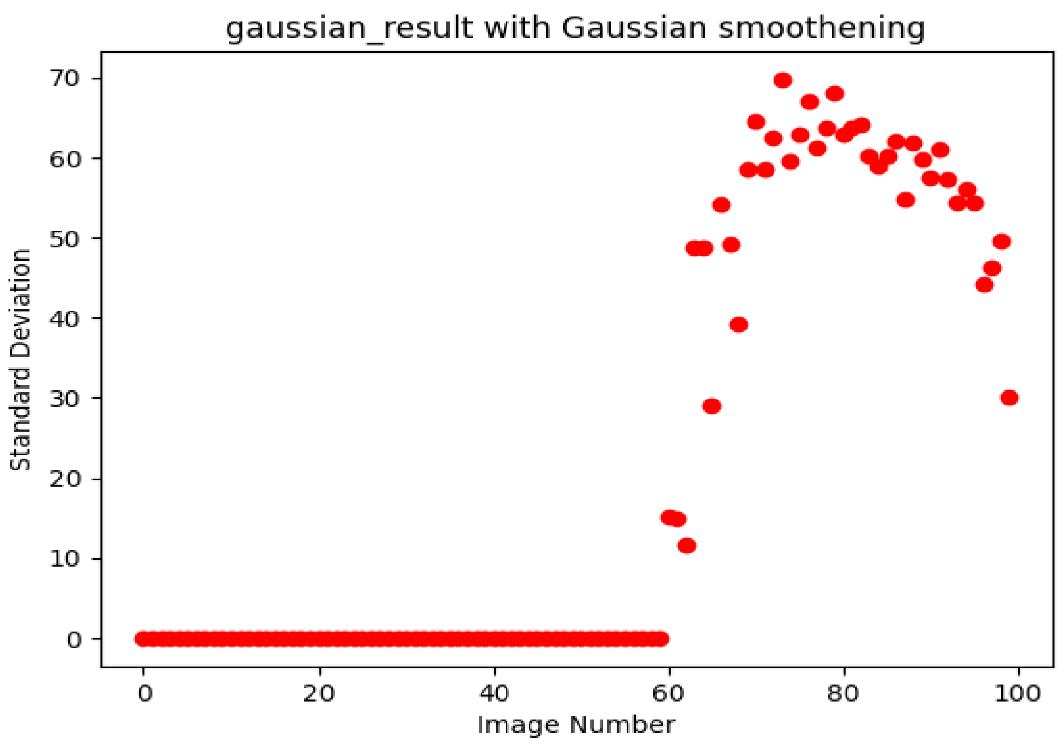
Standard Deviation Plots:





If the images are not smoothened and applied the mask, then the output is not that accurate i.e., motion is detected and there is noise. When there is motion only then the **standard deviation** is **high** for the rest images its **low**.





If the images are smoothened before applying the mask, then the output is very accurate i.e., motion is detected with high accuracy with very less or negligible noise. When there is motion only then the **standard deviation** is **high** for the rest images its **almost zero**.

APPENDIX

GitHub link for the below code

https://github.com/ShivaKumarDande/Projects/blob/main/Computer%20Vision/Motion_Detection.py

```
import os
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

# Function to read the input Images
def read_input_images():
    path = "C:/Users/Shiva Kumar Dande/Downloads/RedChair/RedChair/"
    input_images = []
    #Reading the images from the given path
    #and storing them in a list input_images[]
    for image_Name in os.listdir(path):
        img = cv.imread(os.path.join(path, image_Name))
        if img is not None:
            input_images.append(img)

    #No of images in the list
    n = len(input_images)
    return input_images, n

# Method to convert the images to gray scale
def covert_to_grayScale(images):
    gray_img = []
    for img in images:
        gray_img.append(cv.cvtColor(img, cv.COLOR_BGR2GRAY))
    return gray_img

# Function to apply the operator(1D Differential operator or 1D Gaussian operator)
# on the given images(Gray scaled images or Smoothened Gray scaled images)
def generate_masks(images, operator):
    output_images = [[], [], []]
    mask = []
    #applying differential operator to the gray scaled images
    #output is saved in output_images
    #output_images[0] contains images multiplied with differential_operator[0] = -1/2
    #output_images[1] contains images multiplied with differential_operator[1] = 0
    #output_images[2] contains images multiplied with differential_operator[2] = 1/2
```

```
    for i in range(len(operator)):
        for j in range(len(images)):
            output_images[i].append(operator[i] * images[j])

    #mask[0] = output_images[0][0] + output_images[1][1]  output_images[1][2]
#    num_ops = len(output_images)
#    for j in range(len(output_images[0])-(num_ops-1)):
#        mask.append(output_images[0][j] + output_images[1][j+1] + output_images[2][j+2])

    num_ops = len(output_images)

    for j in range(len(output_images[0])-(num_ops-1)):
        sm = 0
        for i in range(num_ops):
            sm += output_images[i][j+i]
        mask.append(sm)
    mask = np.array(mask)

    return mask

# Applying the threshold on the given masks
# mask values would be 1 if abs(mask) > threshold else 0
def generate_thresholded_masks(mask, threshold):
    mask = 1*(abs(mask) > threshold)
    return mask

# Applying the thresholded mask on the given
# Images (Gray scaled images or Smoothened Gray scaled images)
def apply_thresholded_mask(images, mask, operator):
    #applying the mask to the images
    #mask[0] applied to images[0], images[1], images[2]
    #mask[1] applied to images[1], images[2], images[3]
    result = []
    for i in range(len(mask)):
        for j in range(i, i + len(operator)):
            result.append(mask[i] * images[j])

    result = np.array(result)
#    result_1 = []
#    for i in range(0, len(result)-2, 3):
#        result_1.append(result[0+i] + result[1+i] + result[2+i])

#    result_1 = np.array(result_1)
#    print('size of result is : ')
#    print(np.shape(result))
#    print(np.shape(result_1))
```

```
    return result

# Function to generate 1D derivative of Gaussian with given sigma
def gaussian_derivative(x, sigma):
    A = 1 / (sigma * np.sqrt(2 * np.pi))
    return -A * x * np.exp(-(x**2) / (2 * sigma**2))

def box_filter(images, size):
    images = np.array(images)
    blur_images = cv.blur(images, (size, size))
    return blur_images

def generate_2D_gaussian_filter(ksize, sigma):
    # Create a 2D Gaussian filter
    x, y = np.meshgrid(np.linspace(-1, 1, ksize), np.linspace(-1, 1, ksize))
    d = np.sqrt(x*x + y*y)
    g = np.exp(-(d**2 / (2.0 * sigma**2)))
    # Normalize the filter
    g = g / g.sum()
    return g

def apply_gaussian_2D_filter(images, gaussian_filter):
    images = np.array(images)
    smoothened_images = []
    # Apply the Gaussian filter to each image
    for image in images:
        smoothened_image = cv.filter2D(image, -1, gaussian_filter)
        smoothened_images.append(smoothened_image)
    smoothened_images = np.array(smoothened_images)
    return smoothened_images

def display(image1, image2, waitTime, name):
    display_images = np.concatenate((image1, image2), axis = 2)
#     cv.imshow(name, display_images[140])
#     cv.waitKey(0)
#     cv.destroyAllWindows()
    for i in range(len(display_images)):
        if i % 10 == 0:
            cv.imshow(name, display_images[i])
            cv.waitKey(waitTime)
    cv.destroyAllWindows()

# Function to apply 1D differential Operator on the given
# Images (Gray scaled images or Smoothened Gray scaled images)
def apply_1D_differential_operator(images, differential_operator, threshold):
    # Applying 1D differential operator on the given images
```

```
    differential_operator_mask = generate_masks(images, differential_operator)
    # Applying threshold to the generated masks
    differential_operator_mask_t = generate_thresholded_masks(differential_operator_mask, threshold)
    # Applying the thresholded masks to the images (result)
    differential_operator_result = apply_thresholded_mask(images, differential_operator_mask_t, differential_operator)
    # Converting the type to uint8
    differential_operator_mask = differential_operator_mask.astype(np.uint8)
    differential_operator_mask_t = differential_operator_mask_t.astype(np.uint8)
    differential_operator_result = differential_operator_result.astype(np.uint8)
    return differential_operator_result, differential_operator_mask

def apply_1D_gaussian_operator(images, gaussian_1D_operator, threshold):
    # Applying 1D Gaussian operator on the given images
    gaussian_1D_mask = generate_masks(images, gaussian_1D_operator)
    # Applying threshold to the generated masks
    gaussian_1D_mask_t = generate_thresholded_masks(gaussian_1D_mask,threshold)
    # Applying the thresholded masks to the images (result)
    gaussian_result = apply_thresholded_mask(images, gaussian_1D_mask_t, gaussian_1D_operator)
    # Converting the type to uint8
    gaussian_1D_mask = gaussian_1D_mask.astype(np.uint8)
    gaussian_1D_mask_t = gaussian_1D_mask_t.astype(np.uint8)
    gaussian_result = gaussian_result.astype(np.uint8)
    return gaussian_result, gaussian_1D_mask

def std_deviation(images, title):
    #standard Deviation
    std_dev = []
    std = []
    for i in images:
        std_dev.append(np.std(i))
    x = list(range(0, 100, 1))
    std = std_dev[0:100]

    #plotting std
    plt.title(title)
    plt.xlabel("Image Number")
    plt.ylabel("Standard Deviation")
    plt.plot(x, std, 'o', color ="red")
    plt.show()


images = []
gray_scaled_images = []
```

```
threshold = float(input("Enter the Threshold value : "))
images, n = read_input_images()
print(f'No of Images in the given path is {n}')
gray_scaled_images = covert_to_grayScale(images)

# #display(images, np.zeros(np.shape(images)), 50, 'Original Images and Gray Scaled Images')
# for i in images:
#     cv.imshow('Input Images', i)
#     cv.waitKey(50)
# cv.destroyAllWindows()

# for i in gray_scaled_images:
#     cv.imshow('Gray Images', i)
#     cv.waitKey(50)
# cv.destroyAllWindows()

sigma = float(input('Enter the sigma Value : '))
x = np.linspace(-sigma, sigma, 3)
differential_operator = [-0.5, 0, 0.5]
gaussian_1D_operator = gaussian_derivative(x, sigma)

differential_operator_result, differential_operator_mask =
apply_1D_differential_operator(gray_scaled_images, differential_operator, threshold)
gaussian_result, gaussian_1D_mask = apply_1D_gaussian_operator(gray_scaled_images,
gaussian_1D_operator, threshold)

display(differential_operator_mask, gaussian_1D_mask, 50, 'Differential Mask, Gaussian Mask
(No Smoothening)')
std_deviation(differential_operator_result, "differential_operator_result without smoothening")

display(differential_operator_result, gaussian_result, 50, 'Differential Result, Gaussian Result (No
Smoothening)')
std_deviation(gaussian_result, "gaussian_result without smoothening")

# nr = []
# # concatenate image Horizontally
# for i in range(len(differential_operator_result)):
#     if (i % 3 == 0):
#         nr.append(differential_operator_result[i])

# nr_t = []
# for i in range(len(gaussian_result)):
#     if (i % 3 == 0):
#         nr_t.append(gaussian_result[i])

choice = input("Do you want to smoothen the input Images : \n Enter Y or N : ")
```

```
while (choice == 'Y' or choice == 'y'):
    c = int(input('Choose 1 for Box Smoothening \n Choose 2 for 2D Gaussian Smoothening : '))

    if (c == 1):
        size = int(input('Enter the Size of box filter for Smoothening : '))
        smoothened_images = box_filter(gray_scaled_images, size)
        differential_operator_result, differential_operator_mask = apply_1D_differential_operator(smoothened_images, differential_operator, threshold)
        gaussian_result, gaussian_1D_mask = apply_1D_gaussian_operator(smoothened_images, gaussian_1D_operator, threshold)
        display(differential_operator_mask, gaussian_1D_mask, 50, 'Differential Mask, Gaussian Mask (Smoothening)')
        std_deviation(differential_operator_result, "differential_operator_result with box smoothening")
        display(differential_operator_result, gaussian_result, 50, 'Differential Result, Gaussian Result (Smoothening)')
        std_deviation(gaussian_result, "gaussian_result with box smoothening")

    elif (c == 2):
        ksize = int(input("Enter the 2D Gaussian Filter Size : "))
        ksigma = float(input("Enter the standard deviation of 2D Gaussian Filter : "))
        gaussian_filter = generate_2D_gaussian_filter(ksize, ksigma)
        smoothened_images = apply_gaussian_2D_filter(gray_scaled_images, gaussian_filter)
        display(smoothened_images, gray_scaled_images, 50, 'Smoothened Images, Gray Scaled Images (Smoothening)')
        differential_operator_result, differential_operator_mask = apply_1D_differential_operator(smoothened_images, differential_operator, threshold)
        gaussian_result, gaussian_1D_mask = apply_1D_gaussian_operator(smoothened_images, gaussian_1D_operator, threshold)
        display(differential_operator_mask, gaussian_1D_mask, 50, 'Differential Mask, Gaussian Mask (Smoothening)')
        std_deviation(differential_operator_result, "differential_operator_result with Gaussian smoothening")
        display(differential_operator_result, gaussian_result, 50, 'Differential Result, Gaussian Result (Smoothening)')
        std_deviation(gaussian_result, "gaussian_result with Gaussian smoothening")

    choice = input("Do you want to smoothen the input Images : \n Enter Y or N : ")

print('Thank You!')
```